

---

## 4.5 The master method for solving recurrences

The master method provides a “cookbook” method for solving recurrences of the form

$$T(n) = aT(n/b) + f(n), \quad (4.20)$$

where  $a \geq 1$  and  $b > 1$  are constants and  $f(n)$  is an asymptotically positive function. To use the master method, you will need to memorize three cases, but then you will be able to solve many recurrences quite easily, often without pencil and paper.

The recurrence (4.20) describes the running time of an algorithm that divides a problem of size  $n$  into  $a$  subproblems, each of size  $n/b$ , where  $a$  and  $b$  are positive constants. The  $a$  subproblems are solved recursively, each in time  $T(n/b)$ . The function  $f(n)$  encompasses the cost of dividing the problem and combining the results of the subproblems. For example, the recurrence arising from Strassen's algorithm has  $a = 7$ ,  $b = 2$ , and  $f(n) = \Theta(n^2)$ .

As a matter of technical correctness, the recurrence is not actually well defined, because  $n/b$  might not be an integer. Replacing each of the  $a$  terms  $T(n/b)$  with either  $T(\lfloor n/b \rfloor)$  or  $T(\lceil n/b \rceil)$  will not affect the asymptotic behavior of the recurrence, however. (We will prove this assertion in the next section.) We normally find it convenient, therefore, to omit the floor and ceiling functions when writing divide-and-conquer recurrences of this form.

### The master theorem

The master method depends on the following theorem.

#### Theorem 4.1 (Master theorem)

Let  $a \geq 1$  and  $b > 1$  be constants, let  $f(n)$  be a function, and let  $T(n)$  be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret  $n/b$  to mean either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then  $T(n)$  has the following asymptotic bounds:

1. If  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$ .
2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$ , and if  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$ . ■

Before applying the master theorem to some examples, let's spend a moment trying to understand what it says. In each of the three cases, we compare the function  $f(n)$  with the function  $n^{\log_b a}$ . Intuitively, the larger of the two functions determines the solution to the recurrence. If, as in case 1, the function  $n^{\log_b a}$  is the larger, then the solution is  $T(n) = \Theta(n^{\log_b a})$ . If, as in case 3, the function  $f(n)$  is the larger, then the solution is  $T(n) = \Theta(f(n))$ . If, as in case 2, the two functions are the same size, we multiply by a logarithmic factor, and the solution is  $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$ .

Beyond this intuition, you need to be aware of some technicalities. In the first case, not only must  $f(n)$  be smaller than  $n^{\log_b a}$ , it must be *polynomially* smaller.

That is,  $f(n)$  must be asymptotically smaller than  $n^{\log_b a}$  by a factor of  $n^\epsilon$  for some constant  $\epsilon > 0$ . In the third case, not only must  $f(n)$  be larger than  $n^{\log_b a}$ , it also must be polynomially larger and in addition satisfy the “regularity” condition that  $af(n/b) \leq cf(n)$ . This condition is satisfied by most of the polynomially bounded functions that we shall encounter.

Note that the three cases do not cover all the possibilities for  $f(n)$ . There is a gap between cases 1 and 2 when  $f(n)$  is smaller than  $n^{\log_b a}$  but not polynomially smaller. Similarly, there is a gap between cases 2 and 3 when  $f(n)$  is larger than  $n^{\log_b a}$  but not polynomially larger. If the function  $f(n)$  falls into one of these gaps, or if the regularity condition in case 3 fails to hold, you cannot use the master method to solve the recurrence.

### Using the master method

To use the master method, we simply determine which case (if any) of the master theorem applies and write down the answer.

As a first example, consider

$$T(n) = 9T(n/3) + n.$$

For this recurrence, we have  $a = 9$ ,  $b = 3$ ,  $f(n) = n$ , and thus we have that  $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$ . Since  $f(n) = O(n^{\log_3 9 - \epsilon})$ , where  $\epsilon = 1$ , we can apply case 1 of the master theorem and conclude that the solution is  $T(n) = \Theta(n^2)$ .

Now consider

$$T(n) = T(2n/3) + 1,$$

in which  $a = 1$ ,  $b = 3/2$ ,  $f(n) = 1$ , and  $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$ . Case 2 applies, since  $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$ , and thus the solution to the recurrence is  $T(n) = \Theta(\lg n)$ .

For the recurrence

$$T(n) = 3T(n/4) + n \lg n,$$

we have  $a = 3$ ,  $b = 4$ ,  $f(n) = n \lg n$ , and  $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$ . Since  $f(n) = \Omega(n^{\log_4 3 + \epsilon})$ , where  $\epsilon \approx 0.2$ , case 3 applies if we can show that the regularity condition holds for  $f(n)$ . For sufficiently large  $n$ , we have that  $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$  for  $c = 3/4$ . Consequently, by case 3, the solution to the recurrence is  $T(n) = \Theta(n \lg n)$ .

The master method does not apply to the recurrence

$$T(n) = 2T(n/2) + n \lg n,$$

even though it appears to have the proper form:  $a = 2$ ,  $b = 2$ ,  $f(n) = n \lg n$ , and  $n^{\log_b a} = n$ . You might mistakenly think that case 3 should apply, since

$f(n) = n \lg n$  is asymptotically larger than  $n^{\log_b a} = n$ . The problem is that it is not *polynomially* larger. The ratio  $f(n)/n^{\log_b a} = (n \lg n)/n = \lg n$  is asymptotically less than  $n^\epsilon$  for any positive constant  $\epsilon$ . Consequently, the recurrence falls into the gap between case 2 and case 3. (See Exercise 4.6-2 for a solution.)

Let's use the master method to solve the recurrences we saw in Sections 4.1 and 4.2. Recurrence (4.7),

$$T(n) = 2T(n/2) + \Theta(n),$$

characterizes the running times of the divide-and-conquer algorithm for both the maximum-subarray problem and merge sort. (As is our practice, we omit stating the base case in the recurrence.) Here, we have  $a = 2$ ,  $b = 2$ ,  $f(n) = \Theta(n)$ , and thus we have that  $n^{\log_b a} = n^{\log_2 2} = n$ . Case 2 applies, since  $f(n) = \Theta(n)$ , and so we have the solution  $T(n) = \Theta(n \lg n)$ .

Recurrence (4.17),

$$T(n) = 8T(n/2) + \Theta(n^2),$$

describes the running time of the first divide-and-conquer algorithm that we saw for matrix multiplication. Now we have  $a = 8$ ,  $b = 2$ , and  $f(n) = \Theta(n^2)$ , and so  $n^{\log_b a} = n^{\log_2 8} = n^3$ . Since  $n^3$  is polynomially larger than  $f(n)$  (that is,  $f(n) = O(n^{3-\epsilon})$  for  $\epsilon = 1$ ), case 1 applies, and  $T(n) = \Theta(n^3)$ .

Finally, consider recurrence (4.18),

$$T(n) = 7T(n/2) + \Theta(n^2),$$

which describes the running time of Strassen's algorithm. Here, we have  $a = 7$ ,  $b = 2$ ,  $f(n) = \Theta(n^2)$ , and thus  $n^{\log_b a} = n^{\log_2 7}$ . Rewriting  $\log_2 7$  as  $\lg 7$  and recalling that  $2.80 < \lg 7 < 2.81$ , we see that  $f(n) = O(n^{\lg 7 - \epsilon})$  for  $\epsilon = 0.8$ . Again, case 1 applies, and we have the solution  $T(n) = \Theta(n^{\lg 7})$ .

## Exercises

### 4.5-1

Use the master method to give tight asymptotic bounds for the following recurrences.

a.  $T(n) = 2T(n/4) + 1.$

b.  $T(n) = 2T(n/4) + \sqrt{n}.$

c.  $T(n) = 2T(n/4) + n.$

d.  $T(n) = 2T(n/4) + n^2.$